

# Maintain-e-nator: *Make the World a Better Place*

Xin Liu  
University at Buffalo  
1932 Wallamalo Lane  
xliu36@buffalo.edu

John Longanecker  
University at Buffalo  
P.O. Box 1212  
john.longanecker@gmail.com

Juehui Zhang  
University at Buffalo  
Hekla, Iceland  
larst@affiliation.org

## ABSTRACT

Our system seeks to improve the quality of a facility as well as decrease an organization's overall operating expenses. By letting those who maintain facilities know about issues sooner. Maintenance workers can react quicker and more efficiently if they have better information about the status of their facilities. *Maintain-e-nator* provides a cell phone application to report issues as well as a web interface to allow workers to manage and be notified of issues.

## Categories and Subject Descriptors

H.4 [Information Systems Applications]: Miscellaneous;  
D.2.8 [Software Engineering]: Metrics—*complexity measures, performance measures*

## General Terms

Applications

## 1. INTRODUCTION

Eventually everything will break down. No matter how much money you spend creating a road or building it will run into issues. Buildings start as brand new but will eventually break down. Roads start as smooth, but eventually develop potholes. These breakdowns can sometimes be ignored like a squeaky door, but others can cause safety and health risks. If the stairs of a building are in disrepair they could cause a tripping hazard for other people. The reasons for something not being fixed can vary. Budget, time, priority and awareness all play a role as to when things get repaired. We seek to streamline the process at which maintainers become aware of these issues.

This project is broken down into two components:

1. Android application - This application is used to create a report that will be submitted to a central web server. The user creating this report can submit an image, audio recording and a description of the reported issues.

2. Web Interface - This web interface is broken down into two parts. One is a central place that an authorized user can use to access a list of all submitted issues. This interface allows users to modify an issue such as add additional images or change an issue's status. The other part of the web interface is a public listing of all the currently submitted issues.

So there are two different types of users when it comes to our project:

1. Submitters - These are people who notice or find an issue with the facilities. They will then need to report the issues using the *Maintain-e-Nator* Android application.
2. Maintainers - These are the people who maintain the facility. They will use a web interface to make them aware of an issue that is submitted as well as change a status of an issue.

So we will describe a basic example of how all these examples work together. A submitter is walking through a building on campus and notices that one of the bathroom's automatic faucets are not responding properly when a user moves their hands in front of it. We can take a photo of the faucet and write a short description of the issue. We find that the written explanation and the photo do not accurately describe the problem. So we decide to make a short audio recording of the problem and submit the report.

Once the report is submitted it will show up in the web interface. A maintainer will now log into the authorized section of the web interface. The maintainers can look at issues based on location and date. The Maintainers can now dispatch a worker to go fix the faucet.

This issue is a good example of a problem that may have gone undetected if it were not for the submitter making it known to a maintainer. Most people will be frustrated with a malfunctioning faucet, but will not report this problem if they had to call and report this issue directly to the people who maintain that bathroom.

## 2. MOTIVATIONS

It is easy for a large organization to be unaware of all the maintenance issues that their facilities have. Some obscure room may need a light bulb replaced but those that work in that area do not report the issue or are not around when

the issue is exhibiting its behaviors. So the people who take a night class are the only ones aware of an issues. They do not know where to submit a issue and are often not willing to do the necessary research to find out how to report an issue.

Issues are not limited to the inside of a buildings. They can also involve roads, landscaping, sidewalks, and outdoor sports facilities. For example a pallet could fall off the back of a truck and be sitting in the middle of a road. Most drivers will drive around the pallet and not report it. Many times it is not known how to report an issue like this or it is not convenient to notify the people who maintain the road. Wouldn't it be nice if one could take out their cellphone and submit a report? Our version of *Maintain-e-Nator* is targeted for the University at Buffalo's North Campus, but can easily be configured to be used in any setting.

## 2.1 What is an Issue?

So far we have not defined what a issue is, but have yet to define what constitutes a issue. We consider a maintenance issue to be anything that detracts from the overall quality of a facility. This may seem like a very broad definition for an issues, but the overall range of an issue can vary greatly. At the end of the day the users who submits an issues are the ones who are deciding what an issue is. Who better to determine an issue then those who actually use the facilities?

## 2.2 Goal

The overall goal of this project is to make maintenance workers more aware of the issues that exists on their property. For this project we targeted both indoors and outdoor problems. For indoor issues we only targeted four buildings on the University at Buffalo's North Campus.

The four buildings that we targeted are:

1. Davis
2. Ketter
3. Furnas
4. Jarvis

We only target these four buildings because we did not want to have to large of a scope when it came to using our application. Adding more buildings to our system is not a lot of work, but is not necessary to prove the overall usefulness of this application. If we add more building it will complicated the interface design as well as the testing process of our application.

Buildings are not guaranteed to become better if the maintainers are more aware of issues on there facility. It will certainly help in the process of improving facilities and hopefully in turn create a better facility. We want to make it easier for people to submit problems. The University at Buffalo does provide a phone number for people to call to notify them of building problems, but this process seems dated and not convenient. If you call this number you will be asked to provide some personal information like full name and Student ID number.

## 3. DESIGN

Our application comprises of two parts: the Android application and the backend web application, which are supposed for two kind of users: **submitter** and **maintainer**. Data flow can be depicted as Fig. 1. The data flow of our application is fairly intuitive: whenever submitters spot some facility issues, they open up the Android application, which will try to locate their position and generate the meaningful one, the submitters can take few photos of the issue or record some audio regarding to the issue, then just hit the submit button, that's it! The rest for maintainers, they can use the web application to organize all the submitted issues, log their progresses on repair the issues. And if submitters choose to report the issues with their Google accounts, the backend can also update the issues status to submitters via email notifications automatically.


### 3.1 Android Application

We developed our client application on the Android [1] platform, it is mainly consists of three modules: 1. login module 2. positioning module 3. information collecting module. Below we talk about these three modules in detail.

#### 3.1.1 Login

Whenever a submitter wants to submit an issues they will open the application, and decided to log into the application with their Google account or anonymously. For out examples we will choose a Google account because it is available in (or necessary for ) every Android device. From the users Google account we store the name and email address, and once the user submits some issue, the application will also pass there personal information (just name and email address) to our web application, which can be used in the future for contacting with the user. With the **AccountManager** in the Android library, we can only get the submitter's email address. To make our app more personalized, we want a little more of the submitter's profile information. Once the submitter grants the application permission to retrieve there profile with the access token, we can use it to request the submitter for additional information via calling the **UserInfo** Google API [3]. This whole process follows the **OAuth2** specification [5]. The application doesn't try to remember how the submitter logged in last time. So each time the submitter opens up the app they can choose if they want to remain anonymous or use their Google account.

#### 3.1.2 Positioning

Each time a submitter opens the application and files an issue, the application will initiate a single location update via the Network or GPS. We prefer the Network update over GPS for it is much faster and is accurate enough with the omni Wi-Fi access points within our campus. Since it is unlikely for the submitter to move around when submitting an issue, we only request a *single* location update to save the phones battery. 

Once the application gets the geolocation data, which is a pair of numbers for the latitude and longitude, we need to convert this geolocation into some sort of meaningful information for the submitter. If the submitter is indoors (actually we will get the textual locations for both indoor and outdoor scenarios), the application will try to guess which

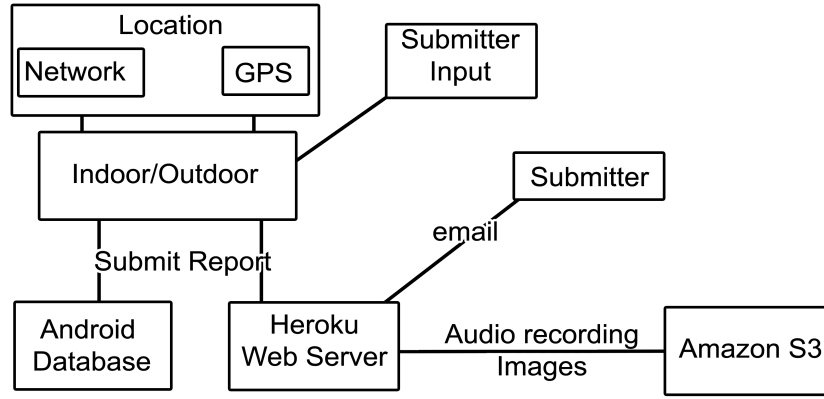


Figure 1:

hall we are currently in and may have to do some calculations using the Euclidean distance between current geolocation and the predefined geolocations of halls we current support, and assume the one with the nearest distance is the hall the submitter is in. Otherwise, if the submitter is outdoors, then the application will try to get a meaningful location of the submitter by using the Google Place API [4] with the geolocation data. Currently we just use the first result returned by this API.

Submitters can modify the location information suggested by the application later. Also, for outdoor scenario, we implemented a **MapView** with a draggable marker which indicates submitter current position. In some cases, if the geolocation provided is not that accurate or submitters move to another place after the location update, they can adjust the position of marker, then the application will use that new geolocation accordingly.

### 3.1.3 Information collecting

For future maintainers to locate the issue position easily and accurately, we hope submitters can report the issue information as detail as possible. Since the location detail for indoor and outdoor could be very different, to ease the process of filling the detail of an issue, the application has two different forms for indoor and outdoor separately, as shown in Fig 2. The application requires submitter to at least provide a description, the location (which maybe acquired automatically as described in Sec. 3.1.2 or replaced by submitter). For current indoor experiment, we only support four halls - Davis Hall, Jarvis Hall, Furnas Hall and Ketter Hall.

Only textual information may not be detail enough when describing an issue, thus we encourage submitter to add photos and also an audio recording regarding the location or issue detail. By long-pressing the image area, the submitter can add up to three photos either from camera capture or gallery. After that the submitter can view the full-size photo by clicking it and decide whether to keep it or replace it with a new one. For the audio recording, to make it act like a two-way radio, we implement it as when submitter holding the recording button, the app will start record the audio until submitter release it. We choose **wav** format for saving the audio recording because it is the quick and dirty way to en-

able the recorded audio be played back both on the Android device and the browser. The supported audio recording formats on Android (to our best knowledge), e.g., **mp4**, **3gp** are not supported by current browsers. While the other formats supported by browsers without any plugins, e.g., **mp3**, **ogg** need third-party native libraries to be recorded on Android devices. For **wav** format, after we collecting the raw audio data, we just need to wrap it up with **wav** header, while the tradeoff is that file size is larger compared to those compressed formats.

### 3.1.4 Others

The application will save all the issues reported by the submitter locally on the Android device if the submitter choose to. Submitter can later view the details of all the issues she reported before. We also add some animations to activity switching, users' interaction to make the application more accessible.

## 3.2 Web Application

For backend web service<sup>1</sup>, we host our web application on **heroku**<sup>2</sup>, which is a cloud application platform that saves us from all kinds of pitfalls in servers management, deployment. Since **heroku** does not store static files as well as users upload files, we use **Amazon S3**<sup>3</sup> for this task.

We use **Django** [2] web framework for our web application, its active community makes it easy and fast to develop web apps with numerous open source modules (a full list of all open source modules we used<sup>4</sup>). Our web application mainly has two components, private admin site, which is for maintainers to manage all the reported issues, and public site, which is for every one to view existing reported issues.

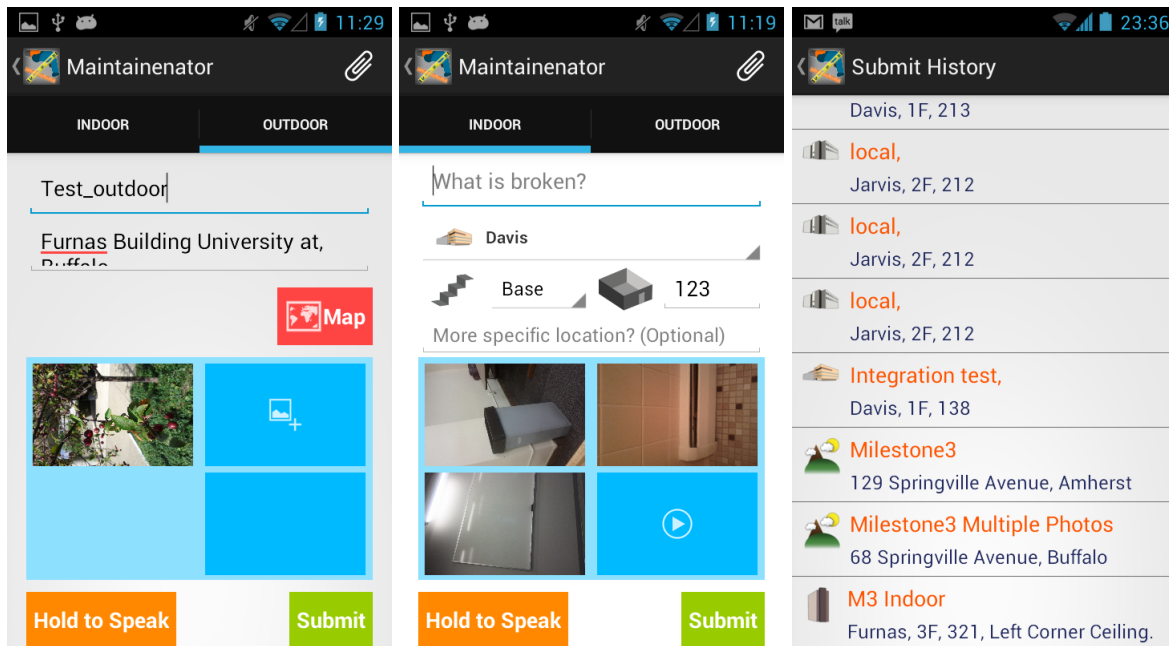
### 3.2.1 Private admin site

<sup>1</sup><http://maintain-e-nator.herokuapp.com>

<sup>2</sup><https://www.heroku.com/>

<sup>3</sup><https://aws.amazon.com/s3/>

<sup>4</sup><https://github.com/forkloop/Maintainenator-backend/blob/master/requirements.txt>



**Figure 2: Android application screenshots, from left to right, outdoor report form, indoor report form, report history.**

Django is already packed with an admin module, which however is not that fancy. We instead use **grappelli**<sup>5</sup> for polishing up the admin interface. The admin site is the main entry for maintainers, which allows multiple maintainers to manage the reported issues, filter them based on the issues location, severity level, status, submitted date *etc.* With the embedded Google Maps, maintainers can get an better idea of where the issues are for outdoor problems. Also, the submitted photos and audio recording can give them more direct intuitions of what the issues are. Maintainers can also keep tracking of their working progress for issues by add logs. Once there is an update for an issue, the web application will send an email notification to the submitter if there is one. Currently, we only send notification to submitter once the issue is fixed.

### 3.2.2 Public site

The web application also has a public site that is intended for all to browse existing issues. Inspired by **Pinterest**, **Airbnb** and other similar websites, we work on this public site to make it as an Single-page application [7], which can offer users a better experience with the infinite-scrolling, no more page reloading when viewing different issues. This is done by adding a model-view-controller (MVC) layer to user browser, and we use **Backbone.js**<sup>6</sup> as our frontend MVC. When user first open this public site, it starts to render a page and fetch the required data asynchronously from web application RESTful API, with user scrolling down for more, it continues pre-fetching more data. Its ability to redraw any part of the UI without requiring a server roundtrip to retrieve HTML makes it more-native-app-like [6]. The whole interaction with the backend is via the web applica-

tion RESTful JSON interface.

### 3.2.3 Others

The web application provides a RESTful API, which the Android application uses to submit an issue, or the public site uses to retrieve the issues information. We use **tastypie**<sup>7</sup> to build this REST-style interface. We did little hack on **tastypie** for this API to support file upload via **mixins**.

Because **heroku** doesn't store user uploaded files, we use **Amazon S3** to store the photos and audio recording uploaded by submitters. With **boot**<sup>8</sup>, we can achieve the transfer of uploaded files from **Heroku** to **Amazon S3** transparently, once the web application receives a file from submitter, it is written to **Amazon S3** directly.

Normally, for a web application, besides handling income requests, it has other tasks like preprocessing uploaded images, sending emails to registered users, and you don't want to mix these kinds of slow processes with HTTP handlers, for this would greatly slow down the request-response cycle, jeopardize users experience. Thus we separate these kinds of tasks to other processes, and use **foreman**<sup>9</sup> to manage them all. There are two kinds of processes for our web application, the **web process** is for responding all HTTP requests, either rendering a view or returning some JSON data. While the **worker process** is for all other not emergent tasks, *e.g.*, sending submitters emails regarding the issues they submitted.

## 4. TEST

<sup>7</sup><http://tastypieapi.org/>

<sup>8</sup><http://docs.pythonboto.org/>

<sup>9</sup><http://ddollar.github.com/foreman/>

<sup>5</sup><http://www.grappelliproject.com/>

<sup>6</sup><http://backbonejs.org/>

We developed and tested our Android application on Google Nexus S with 4.1.2 version. The Android application should also work fine on the 4.0.\* platform. The textual location suggested by Google Places API would be less accurate for outdoor place near the new building, e.g., Davis Hall (possible reason could be that this new building hasn't been indexed by Google). The size of the `wav` file for a 10 seconds audio recording is around 1.8M. And normally, it takes 4 seconds to upload an issue report having two photos and a 5 seconds audio recordings with Wi-Fi connected. Source codes for both Android application<sup>10</sup> and web application<sup>11</sup> are available on `GitHub`.

## 5. FUTURE WORK

Though our application . . . , there are a lot room for future improvement. (i) Current we didn't compress the photos and audio recording when sending them to the web application, which maybe consume a lot of data if the phone doesn't connect to Wi-Fi. To fix this, we can resize down the photos, and use a compressed audio format for recording the audio. Or we can allow submitters save current report and submit it later when they have Wi-Fi connection. (ii) The web application admin site is still . . . , some functions like collapse the same issues reported by different submitters, allow issues fixing assignment to maintainers. Meanwhile, we would like to use some open-source project tracking system, e.g., `trac`<sup>12</sup>, and tailoring as we need. (iii) We only tested the application within three of us, in the future, we can utilize `PhoneLab`<sup>13</sup> to fully test our application.

## 6. CONCLUSIONS

Maintenance issues have been an annoying problem for many years. People may find some issues nearby but they don't know how to report them. We propose a new android application called `Maintain-e-Nator`, to make it easy for people to report their issues. People can choose indoor or outdoor, write down the description of problem, take at most three pictures, and even do some recording. We also build a web application, which is used by maintainers to manage the reported issues, filtering them based on the location, severity level, status, submitted date etc. Maintainers can view the issues locations on Google Maps, get an idea of what the issues are via submitted photos and audio recording.

## 7. ACKNOWLEDGMENTS

We would like to thank Prof. Steven Y. Ko and Prof. Geoffrey Challen for their valuable suggestions. We would also like to thank `PhoneLab` for providing us Android phones.

## 8. REFERENCES

- [1] Android. <http://www.android.com/>.
- [2] Django. <https://www.djangoproject.com/>, 2012.
- [3] Google. <https://developers.google.com/accounts/docs/oauth2>.
- [4] Google-Places-API. <https://developers.google.com/places/documentation/>.

- [5] O. W. Group. The oauth 2.0 authorization framework. 2012.
- [6] Mixu. Single page apps in depth, 2012.
- [7] Wikipedia. Single-page application — wikipedia, the free encyclopedia, 2012.

<sup>10</sup><https://github.com/forkloop/Maintain-e-nator>

<sup>11</sup><https://github.com/forkloop/Maintainenator-backend>

<sup>12</sup><http://trac.edgewall.org/>

<sup>13</sup><http://www.phone-lab.org/>

## APPENDIX

### A. ACCOMPLISHMENTS

Here we list our accomplishments for each milestone, the slides for each milestone demo can be viewed in **speakerdeck**<sup>14</sup>.

#### A.1 Milestone 1

- Set up the web application on **heroku**, has a simple public site.
- Develop the first version of Android application which is able to send some textual information and one photo to backend web application.

#### A.2 Milestone 2

- Rewrite **FormActivity** with **Fragment**, separate indoor and outdoor information form.
- Tweak on Android application, handle special scenarios when there is no location update, or network connection.
- Separate photo from issue model, allows multiple photos upload.
- Allow submitter to save submitted issues.
- Add admin site to web application.

#### A.3 Milestone 3

- Fully UI polishing for Android application, with designed icons, image buttons, new login activity, etc.
- Add audio recording function, allow submitter to also submit a short audio.
- Add a **MapView** to outdoor for submitter to adjust their location with Google Map interactively.
- Rewrite public site as Single-page application.

---

<sup>14</sup><https://speakerdeck.com/forkloop/cse622-milestone-reports>